Optimal Allocation of Testing Resource for Modular Software based on Testing-Effort Dependent Software Reliability Growth

N. Ahmad[1], M.G.M. khan[2] and Syed Faizul Islam[1]
[1]University Department of Statistics and Computer Applications
T. M. Bhagalpur University, Bhagalpur-812007, India
Emails: nesar_bgp@yahoo.co.in; syed123468@gmail.com
[2]School of Computing Information and Mathematical Sciences
The University of the South pacific, Suva, Fiji Island
Email: Khan_mg@usp.ac.fj

***Abstract*. Software reliability is a key factor in software development process. Testing phase of software begins with module testing whereby, modules are tested independently to remove substantial amount of faults within a specified testing resource. Therefore, the available resource must be allocated among the modules in such a way that number of faults is removed as much as possible from each of the module to achieve higher software reliability. In this paper two optimization problem are discussed for optimal allocation of testing resources for the modular software system. These optimization problems are formulated as nonlinear programming problems (NLPP), which are modeled by a software reliability growth model based on a non-homogeneous Poisson process which incorporated Log-logistic testing-effort function. LINGO program is used to solve the optimization problems. Finally, numerical examples are given to illustrate the procedure developed in this paper. It is shown that the optimal allocation of testing-resources among software modules can improve software reliability.**

***Keywords*: Software reliability growth model, resource allocation problem, nonlinear programming problem, inflection S-shaped models.**

I. INTRODUCTION

Computers and software together has changed the way we live, trade, explore and enjoy life for the better. Software is broadly classified as operating system and applications software. Software plays a vital role in the modern life. Software is a functioning element embedded in computers. The computers are being almost used everywhere, like medical fields, businesses, chemical labs, air traffic control towers, ships, space ships, home appliances, communication, manufacture and many more. Hence, relying on machines and its reliability has been increased. Software reliability becomes a crucial feature of the computer systems. Therefore, the breakdown in the system could result in the fiscal, possessions, and human loss. A malfunctioning pacemaker in a heart patient or a Mars path finder that has lost contact due to a software bug or a hacker taking advantage of a bug in financial system to withdraw away cash electronically portrays the problem.

A computer system consists of two major components: hardware and software. Software is a created by man therefore, a high degree of reliability cannot be guaranteed. Thus, researches have been conducted over the past decades to study the software reliability and many software reliability growth models (SRGM) have been proposed (Lyu (1996) & Musa *et al*., 1987). Software reliability is defined as the probability of failure-free operation of a computer program for a specified time in a specified environment (Musa *et al*., 1987). Software is considered to have performed a successful operation, when it functions completely as expected, without any failure. Hence, software reliability is a key factor in software development process.

A software development process consists of four phases: specification, design, coding and testing (Myers, 1976). It is the final phase where the software is tested to detect and correct software faults in the following three consecutive stages: module testing, integration testing, and system testing. Software consists of several modules that are tested independently during the module testing, and the interfaces among these modules are tested in the integration testing. Lastly, the complete software system of which modules are interconnected is tested under the stimulated user environment in the system testing. Moreover, all the testing activities of different modules should be completed approximately 40% - 50% of the total amount of software development resources (Yamada *et al*., 1995). Hence project managers should be able to effectively allocate the testing resources among all the modules to develop highly reliable software. Many papers have addressed the optimal resource allocation problem over the years, including Ohtera and Yamada (1990);

Yamada *et al*. (1995); Huang *et al*. (2002 & 2004); Huang and Lyu (2005); Huang and Lo (2006), Lo *et al.* (2002); Lyu *et al*. (2002); Leung (1997); Kapur *et al*. (2004); Khan *et al*. (2008); and Xie and Yang (2001).

This paper discusses a management problem to achieve a reliable software system efficiently during module testing in the software development process by applying software reliability growth model. In order to develop a quality and reliability software system, it is very important for the manager to allocate the specified amount of testing-resource expenditures in the module testing in terms of time-dependent behavior of the cumulative number of faults detected during the testing phase.

In this paper, two strategies of optimal resource allocation problems for module testing are discussed and formulated as nonlinear programming problems (NLPP). The first NLPP maximizes the total number of faults expected to be removed during module testing satisfying the available testing resources. Sometimes management may want a desire level of reliability to be achieved or/and a certain percentage of number of faults to be removed. Adding these requirements to the first NLPP as constraints the other NLPPs are formulated. These NLPPs are modeled by software reliability growth model based on a non-homogeneous Poisson process (NHPP) with Log-logistic testing-effort function. Both problems are mathematical programming problem. These are solved using the LINGO program. The methodology discussed in the paper has been illustrated through numerical examples.

Notations

N: Number of modules in the Software (>1)

$a(a_i)$: Expected number of faults in the software (ith module; i = 1,2,…N)

$b(b_i)$: Proportionality constant (for the ith module)

$r(r_i)$: Fraction of independent faults (for the ith module)

$x(t)$ $(x_i(t))$: Current testing effort expenditure at testing time t,

$$X(t) = \int_0 x(w)dw, \ (X_i(t) = \int_0 x_i(w)dw)$$

X_i, Z: The amount of testing resources to be allocated to the ith module and total testing resource available

$m(t)$ $(m_i(t))$: Number of faults removed in (0, t] (for the i = 1,2,…N)

T: Total testing time

X_i^*: Optimal value of X_i (i = 1, 2,… N)

l_i: desired level of reliability level (i.e. number of faults desired to be removed from the ith module)

II. SOFTWARE RELIABILITY GROWTH MODEL FOR MODULES

A. Classification:

NHPP : non-homogeneous Poisson process

Testing-Resource: resource expenditure spent on software testing, e.g., man-power, CPU hours, executed test-cases

Fault : a cause of a failure which is unacceptable departure from nominal program operation

Module testing: verification a single software module in a simulated software environment where the module is isolated from all other modules.

A software reliability growth model (SRGM) is a mathematical expression of the software error occurrence and the removal process. A software reliability growth model explains the time dependent behavior of fault removal. In this paper a resource allocation problem is discussed using such a SRGM for the modules. Numerous SRGMs have been developed during the last three decades and they can provide very useful information about how to improve reliability (Musa *et al.*, 1987; Xie, 1991; Lyu, 1996). Among these models, exponential growth model and inflection S-shaped growth model have been shown to be very useful in fitting software failure data. Many authors incorporated the concept of testing-effort into exponential type SRGM based on the NHPP to get a better description of the fault detection phenomenon.

The inflection S-shaped NHPP software reliability growth model is known as one of the flexible SRGMs that can depict both exponential and S-shaped growth curves depending upon the parameter values (Ohba, 1984 & Kapur *et al.*, 2004) .The model has been shown to be useful in fitting software failure data. Ohba proposed that the fault removal rate increases with time and assumed the presences of two types of errors in the software. Later, Kapur *et al.* (2004), Khan *et al.* (2008) and Ahmad *et al.* (2010; 2010a; 2011) modified the inflection S-shaped model and incorporated the testing-effort in an NHPP model.

The extended inflection S-shaped SRGM with Log-logistic testing-effort function is formulated on the

following assumptions (Ohba, 1984; 1984a; Yamada and Osaki, 1985; Yamada *et al.*, 1986; 1993; Kapur *et al.*, 1999; Kuo *et al.*, 2001; Huang and Lo, 2006; Kapur *et al.*, 2004; Ahmad *et al.*, 2010; 2011):

1) The software system is subject to failures at random times caused by errors remaining in the system.
2) Error removal phenomenon in software testing is modeled by Non Homogeneous Poisson Process (NHPP).
3) The mean number of errors detected in the time interval $(t, t+\Delta t]$ by the current testing-effort expenditures is proportional to the mean number of detectable errors in the software.
4) The proportionality increases linearly with each additional error removal.
5) Testing-effort expenditures are described by the Log-logistic testing-effort.
6) Each time a failure occurs, the error causing that failure is immediately removed and no new errors are introduced.
7) Errors present in the software are of two types: mutually independent and mutually dependent.

The mutually independent errors lie on different execution paths, and mutually dependent errors lie on the same execution path. Thus, the second type of errors is detectable if and only if errors of the first type have been removed. According to these assumptions, if the error detection rate with respect to current testing-effort expenditures is proportional to the number of detectable errors in the software and the proportionality increases linearly with each additional error removal, we obtain the following differential equation:

$$\frac{(d/dt)m(t)}{x(t)} = \phi(t)(a - m(t))$$

(2.1)

where

$$\phi(t) = b\left[r + (1-r)\frac{m(t)}{a}\right],$$

Solving Equation (2.1) with the initial condition that, at $t = 0,\ X(t) = 0,\ m(t) = 0$, the following is obtained:

$$m(t) = \frac{a\left[1 - e^{-bX(t)}\right]}{1 + (1-r)/r\ e^{-bX(t)}}$$

(2.2)

To describe the behavior of testing effort, Log-logistic function (Bokhari and Ahmad, 2006; Ahmad *et al.*, 2010a; 2011) has been used.

The Cumulative testing-effort expenditure consumed in (*0, t*] is depicted in the following:

$$X(t) = \alpha[1 - \{1 + (\beta t)^{\theta}\}^{-1}]$$

$$= \alpha[(\beta t)^{\theta} / (1 + (\beta t)^{\theta})], \alpha > 0,\ \beta > 0, \theta > 0, t > 0\ .$$

(2.3)

and the current testing-effort consumed at testing time t is

$$x(t) = [\alpha\beta\theta(\beta t)^{\theta-1}] / [1 + (\beta t)^{\theta}]^2\ ,$$

(2.4)

where α, β, and θ are constant parameters, α is the total amount of testing-effort expenditures; β is the scale parameter, and θ are shape parameters

The software reliability representing the probability that no failures occur in the time interval $(t, t+\Delta t)$ given that the last failure occurred at testing time Δt, is given by

$$R(\Delta t \mid t) = e^{-\ m(t+\Delta t) - m(t)}$$

Taking the logarithm on both sides of the above equation, we obtain

$$\ln R(\Delta t \mid t) = -\ m(t + \Delta t) - m(t)\ \ .$$

Now, we define another measure of reliability as a ratio of cumulative number of detected errors at time *t* to the expected number of initial errors (Huang and Lyu, 2005; Huang and Lo, 2006).

$$R(t) \equiv \frac{m(t)}{a} = \frac{\left[1 - e^{-bX(t)}\right]}{1 + ((1-r)/r)e^{-bX(t)}}$$

B. Estimation of Parameters

Using the method of Least square estimation, the parameters α, β, and θ in Log-logistic testing-effort function can be estimated. These parameters are determined for *n* observed data pairs in the form $(t_k,\ X_k)\ (k = 1, 2, ..., n; 0 < t_1 < t_2 < .... < t_n)$, where X_k is the cumulative testing-effort consumed in time $(0, t_k]$. The least square estimators $\hat{\alpha}$, $\hat{\beta}$, and $\hat{\theta}$ can be estimated by minimizing the following:

$$\text{Minimize}\ \ \sum_{i=1}^{n}\left[X_i - \hat{X}\right]$$

Subject to $\hat{X} = X_n$ (i.e. the estimated value of testing effort is equal to the actual value).

Once the estimates α, β, and θ are known, the parameters $a, b, \text{and } r$ of the SRGMs for the module can be estimated through Maximum Likelihood Estimation method. The estimators of $a, b, \text{and } r$ are determined for the n observed data pairs in the form $(t_k, y_k)\ (k = 1, 2, ..., n;\ 0 < t_1 < t_2 < ..... < t_n)$ where y_k is the cumulative number of software errors detected up to time t_k or $(0, t_k]$. The likelihood function for the module is given by:

$$L(a,b,r/(y,X) = \prod_{k=1}^{n} \frac{m(t_k) - m(t_{k-1})^{(y_k - y_{k-1})}}{(y_k - y_{k-1})!} . e^{-[m(t_k) - m(t_{k-1})]}$$

where $t_0 = 0$ and $y_0 = 0$

III. TESTING-RESOURCE ALLOCATION PROBLEM

In module testing two kinds of testing-resource allocation problems are considered to make the best use of a specified total testing-resource allocation. It has to be allocated appropriately to each software module which is tested independently and simultaneously. In this section, we will consider a resource allocation problems based on an SRGM with Log-logistic testing-effort function during software testing phase.

1) Assumptions (Xie, et al., 2001; Huang and Lyu, 2005; Huang and Lo, 2006; Yamada *et al.*, 1995; Khan *et al.* 2008):

2) The software system is composed of *N* independent modules, and the software modules are tested independently. The number of software faults remaining in each module can be estimated by the model.
3) The total amount of testing-resource expenditures for the module testing is specified.
4) The manager has to allocate the specified total testing-resource expenditures to each software module so that the software faults to be removed in the system may be maximized. The desired software reliability level is achieved.

The length of module testing is often fixed when scheduling is done for the whole testing phase. Therefore, limited resources are available, which need to be allocated wisely. The first optimization problem in view of the model from section 2 with *N* modules can be formulated as below, whereby m_i faults are expected to be removed from the ith module with effort X_i. This problem considers the resource constraint only. In the second optimization problem discussed in this paper incorporates an additional constraint to the basic model.

$$\text{Maximize } \sum_{i=1}^{N} m_i(t) = \sum_{i=1}^{N} \frac{a_i(1 - e^{-b_i X_i(t)})}{1 + [(1 - r_i)/r_i] e^{-b_i X_i(t)}}$$

$$\text{Subject to } \sum_{i=1}^{N} X_i \le Z$$

$$X_i \ge 0, \quad i = 1, ..., N$$

(P1)

The total error content in the software is calculated from the estimates of parameters of SRGMs for modules. With the availability of the resources, module testing targets at removing maximum numbers of faults. The important concern here is to how much to test. Therefore, the software testing time (T) is almost fixed. Let X_i be the testing effort that has to be spent on the ith module during testing time T, the mean value function can be then written as

$$m_i(X_i) = \frac{a_i(1 - e^{-b_i X_i})}{1 + [(1 - r_i)/r_i] e^{-b_i X_i}}, \quad i = 1, ..., N$$

Hence, the problem (P1) can be restated as follows

$$\text{Maximize } \sum_{i=1}^{N} m_i(X_i) = \sum_{i=1}^{N} \frac{a_i(1 - e^{-b_i X_i})}{1 + [(1 - r_i)/r_i] e^{-b_i X_i}}$$

$$\text{Subject to } \sum_{i=1}^{N} X_i \le Z$$

$$X_i \ge 0, \quad i = 1, ..., N$$

(P2)

The above optimization problem (P2) can be solved by developing a program in LINGO.

Sometimes some of the modules may not get any resources in the above allocation procedure, to which the management may not agree where one or more modules are not tested any further. During module testing, it is expected that each module is tested satisfactorily so that a certain percentage of the error content is desired to be removed in each module of the software. That is, to aspire certain reliability level for the software as well as for each of the modules. Consequently, the above optimization problem (P2) needs to be modified to maximize the removal of the faults in the software under resource constraint and the desired level of faults to be removed (to achieve desired level of reliability) from each of the

modules in the software. Therefore, the resulting resource allocation problem can be stated as follows:

$$\text{Maximize} \sum_{i=1}^{N} m_i = \sum_{i=1}^{N} \frac{a_i (1 - e^{-b_i X_i})}{1 + \delta_i e^{-b_i X_i}}, \quad \text{where } \delta_i = [(1 - r_i)/r_i]$$

$$\text{Subject to } m_i = \frac{a_i (1 - e^{-b_i X_i})}{1 + \delta_i e^{-b_i X_i}} \geq l_i a_i$$

$$\sum_{i=1}^{N} X_i \leq Z$$

$$X_i \geq 0, \quad i = 1, \ldots, N$$

(P3)

Thus, from

$$\frac{a_i (1 - e^{-b_i X_i})}{1 + \delta_i e^{-b_i X_i}} \geq l_i a_i$$

Therefore,

$$X_i \geq -\frac{1}{b_i} \ln\left(\frac{1 - l_i}{1 + l_i \delta_i}\right)$$

(3.1)

Hence, the above optimization problem (P3) can be solved by developing a program in the LINGO.

IV. NUMERICAL EXAMPLE

In this section, we discuss a example to demonstrate the use of proposed method and to show how the optimal allocation of testing resource to each module is determined. We assume that parameters a_i, b_i, and r_i $(i = 1, 2, \ldots, 8)$, for a software system consisting of eight modules, have already been estimated by MLE using the software failure data and are summarized in Table 1 (Kapur *et al*., 2004; Khan *et al*., 2008). The total resource taken for the testing is 110,000 units.

The problem (P1) through problem (P2) is solved by developing a program as stated below using LINGO program.

```
max=(45*(1-2.718281828^(-
0.000412932*x1)))/(1+0.170795673*(2.718281828^(-
0.000412932*x1)))+(13*(1-2.718281828^(-
0.000319987*x2)))/(1+0.129642*(2.718281828^(-
0.000319987*x2)))+(16*(1-2.718281828^(-
0.000264216*x3)))/(1+0.124116*(2.718281828^(-
0.000264216*x3)))+(35*(1-2.718281828^(-
0.000150112*x4)))/(1+0.265847*(2.718281828^(-
0.000150112*x4)))+(14*(1-2.718281828^(-
0.0000895707*x5)))/(1+0.256627*(2.718281828^(-
0.0000895707*x5)))+(21*(1-2.718281828^(-
0.0000587323*x6)))/(1+0.32464*(2.718281828^(-
0.0000587323*x6)))+(20*(1-2.718281828^(-
0.0000320165*x7)))/(1+0.697118*(2.718281828^(-
0.0000320165*x7)))+(11*(1-2.718281828^(-
0.0000315115*x8)))/(1+0.728208*(2.718281828^(-
0.0000315115*x8)));

Subject to

x1+x2+x3+x4+x5+x6+x7+x8<=110000;

x1, x2 ,x3, x4, x5, x6, x7, x8>0;
```

The optimal allocation of resources X_i for the modules are computed using the LINGO program is summarized in the Table 1 along with the corresponding expected number of removed, percentages of removed and remaining fault for each module.

Table 1

Module	a_i	b_i	R_i	X_i *	m_i	% of fault removed	% of fault remained
1	45	0.000413	0.8541	10255.11	44	98	2
2	13	0.000319	0.8852	8405.585	12	92	8
3	16	0.000264	0.8895	10225.36	15	93	7
4	35	0.000150	0.7899	20178.14	33	94	6
5	14	8.95E-05	0.7957	16787.11	10	74	26
6	21	5.87E-05	0.7549	25753.15	15	73	27
7	20	3.20E-05	0.5892	18395.53	6	32	68
8	11	3.15E-05	0.5786	0	0	0	100
Total	175			110000	136	78	22

The total number of faults removed from the software through this allocation is 136. That is, 78 % of the fault content being removed. It is also noted that in module 7, the remaining fault after allocation is higher than the fault removed and in module 8 none of the faults are removed. Hence, there could be regular malfunction in

the operational phase which is not desired. A reliability level of 50% for each module is usually desired by the software developers (i.e., $l_i = 0.5,\ i = 1,.....,8$). Therefore, solving the problem (P3) through equation (3.1), a new allocation of resources is obtained. This result along with the expected number of faults removed, percentages of faults removed, and faults remaining for each module are given in the Table 2. The total number of faults that can be removed through this allocation is 118 (i.e. 68% of the total fault content is removed).

In addition to above, a different reliability level (i.e. percentage of fault to be removed from each module can be varied) can be achieved for different modules in a software (i.e. by varying the values of l_i). Further tests also can be done whereby the developer desires to remove certain percentage of the total fault content with an increase in the total resources for the testing.

Table 2

Module	a_i	l_i	b_i	R_i	X_i*	m_i	% of fault removed	% of fault Remained
1	45	0.5	0.000413	0.854	6546.94	42	92	8
2	13	0.5	0.000319	0.885	3469.03	8	64	36
3	16	0.5	0.000264	0.889	4261.16	10	65	35
4	35	0.5	0.000150	0.789	9432.94	25	71	29
5	14	0.5	8.95E-05	0.796	9086.35	7	50	50
6	21	0.5	5.87E-05	0.755	14362.8	10	50	50
7	20	0.5	3.20E-05	0.589	30990.5	10	50	50
8	11	0.5	3.15E-05	0.578	31850.1	6	50	50
Total	175				110000	118	68	32

V. CONCLUSION

This paper developed the strategies on the optimal testing resource allocation problems for modular software system. We discussed numerical example on resource allocation problems for module testing to show the applications and impacts of proposed method. Based on the experimental results, we conclude that the proposed strategies may be helpful to software project managers for making the best decisions in solving these problems.

VI. REFERENCES

[1]. Ahmad, N., Khan, M. G. M and Rafi, L. S. (2011), "Analysis of an Inflection S-shaped Software Reliability Model Considering Log-logistic Testing-Effort and Imperfect Debugging", *International Journal of Computer Science and Network Security*, Vol. 11 (1), pp. 161 – 171.

[2]. Ahmad, N., Khan, M. G. M and Rafi, L. S. (2010), "A Study of Testing-Effort Dependent Inflection S-Shaped Software Reliability Growth Models with Imperfect Debugging", International Journal of Quality and Reliability Management, Vol. 27 (1), 89-110.

[3]. Ahmad, N., Khan, M. G. M and Rafi, L. S. (2010a), "Software Reliability Modeling Incorporating Log-Logistic Testing-Effort with Imperfect Debugging", in Proceedings of the International Conference on Modeling, Optimization and Computing (ICMOC-2010), Durgapur, India, Published by American Institute of Physics, pp. 651-657.

[4]. Bokhari, M.U. and Ahmad, N. (2006), "Analysis of a software reliability growth models: the case of log-logistic test-effort function", In: Proceedings of the 17th IASTED International Conference on Modeling and Simulation (MS'2006), Montreal, Canada, pp. 540-545.

[5]. Lo, J.H., Kuo, S.Y., Lyu, M.R. and Huang, C.Y., (2002), "Optimal Resource Allocation and Reliability analysis for Component-based Software Applications", Proceedings of the 26th IEEE Annual International Computer Software and Applications Conference (COMPSAC'2002), pp.7-12.

[6]. Huang, C.Y., Lo, J.H., Kuo, S.Y. and Lyu, M.R. (2002), "Optimal Allocation of Testing Resources for Modular Software Systems", Proceedings of the Thirteenth IEEE International Symposium on Software Reliability Engineering, pp. 129-138.

[7]. Huang, C.Y., Lo, J.H., Kuo, S.Y. and Lyu, M.R. (2004), "Optimal Allocation of Testing Resource Considering Cost, Reliability, and Testing-Effort", Tenth IEEE Pacific Rim International Symposium on Dependable Computing, pp. 103-112.

[8]. Huang, C.Y., and Lyu, M.R., (2005), "Optimal Testing Resource Allocation, and Sensitivity Analysis in Software Development", IEEE Transaction on Reliaility, Vol. 54, No. 4, pp. 592-603.

[9]. Huang, C.Y. and Lo, J.H. (2006), "Optimal Resource Allocation for Cost and Reliability of Modulare Software Systems in the Testing Phase", The Journal of Systems and Software, Vol. 79, Issue 5, pp. 653-664.

[10]. Khan, M. G. M., Ahmad, N., and Rafi, L. S. (2008), "Optimal Testing Resource Allocation for Modular Software Based on a Software Reliability Growth Model: a Dynamic Programming Approach", in *Proceedings of the International Conference on Computer Science and Software Engineering (CSSE-2008)*, Wuhan, China, *IEEE Computer Society*, pp. 759-762.

[11]. Kapur, P.K., Jha, P.C., and Bardhan, A.K. (2004), "Optimal Allocation of Testing Resource for a Modular Software", Asia-Pacific Journal of Operational Research, Vol. 21, no. 3, pp. 333-354.

[12]. Kapur P.K., Garg R.B. and Kumar S. (1999), Contributions to Hardware and Software Reliability, World Scientific, Singapore.

[13]. Kuo, S.Y., Huang, C.Y. and Lyu, M.R. (2001), "Framework for Modeling Software Reliability, using Various Testing-Efforts and Fault-Detection Rates", IEEE Transitions on Reliability, Vol. 501, No. 3, pp. 310-320.

[14]. Leung, Y.W. (1997), "Dynamic Resource Allocation for Software Module Testing", *Journal of Systems Software*, Vol. 37, 129-139.

[15]. Lyu, M.R. (1996), Handbook of Software Reliability Engineering, McGraw Hill.

[16]. Lyu, M.R., Rangarajan, S., and Van Moorsel, A.P.A. (2002), "Optimal Allocation of Test Resources for Software Reliability Growth Modeling in Software Development", IEEE Transactions on Reliability, Vol. 51, No. 2, pp. 183-192.

[17]. Musa, J.D., Iannino, A. and Okumoto, K. (1987), Software Reliability: Measurement, Prediction and Application, McGraw-Hill.

[18]. Myers, G.J. (1976), Software Reliability: Priniples and Practices, John Wiley & Sons.

[19]. Ohba, M. (1984), "Software Reliability Analysis Model", IBM Journal Res. Development, Vol. 28, No. 4, pp. 428-443.

[20]. Ohtera, H. and Yamada, S. (1990), "Optimal Allocation and Control Problems for Software-Testing Resources", IEEE Transactions on Reliability, Vol. 39, no. 2, pp.171-176.

[21]. Xie, M. (1991), Software Reliability Modeling, Singapore: World Scientific.

[22]. Xie, M. and Yang, B. (2001), "Optimal testing-Time Allocation for Modular Systems", International Journal of Quality and Reliability Management, Vol. 18, No. 8, pp. 854-863.

[23]. Yamada, S. and Osaki, S. (1985), "Software Reliability Growth Modeling: Models and Applications", IEEE Transactions on Software Engineering, Vol. Se.11, No. 12, pp. 1431-1437

[24]. Yamada, S., Ohtera, H. and Narihisa, H. (1986), "Software Reliability Growth Models with Testing-Effort", IEEE Transactions on Reliability, Vol.R.35, No.1, pp. 19-23.

[25]. Yamada, S., Hishitani, J. and Osaki, S. (1993), "Software Reliability Growth with a Weibull test-Effot: A Model & Application", IEEE Transactions on Reliability, Vol.42, No.1, pp. 100-105.

[26]. Yamada, S., Ichimori, T. and Nishiwaki, M. (1995), "Optimal Allocation Policies for Testing-Resource Based on a Software Reliability Growth Model", International Journal of Mathematical and Computer Modeling, Vol. 22, No. 10-12, pp. 295-301.